Branch & Bound (B&B) and Constraint Satisfaction Problems (CSPs)

Alan Mackworth

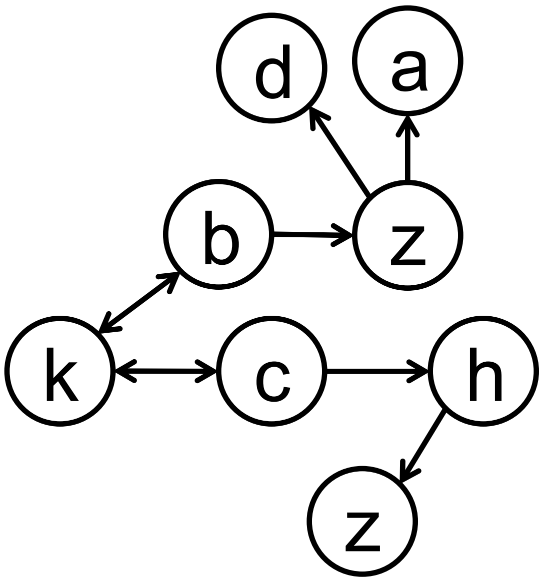
UBC CS 322 – CSP 1
January 25, 2013

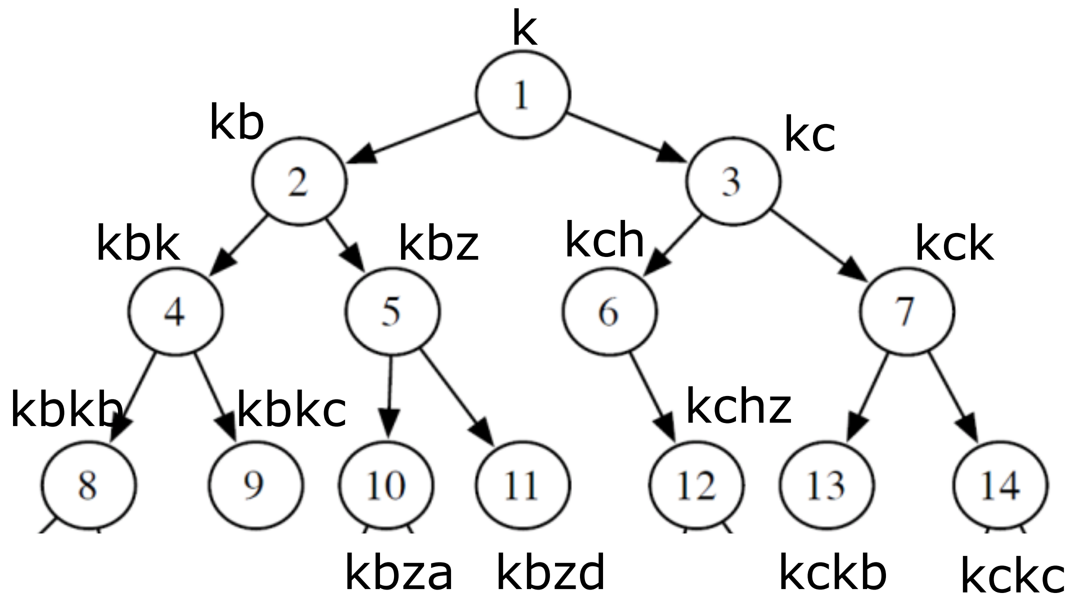
P&M textbook § 3.7.4 & 4.0-4.2

Lecture Overview

- Recap
- Branch & Bound
- Wrap up of search module
- Constraint Satisfaction Problems (CSPs)

Recap: state space graph vs search tree





State space graph.

Search tree.

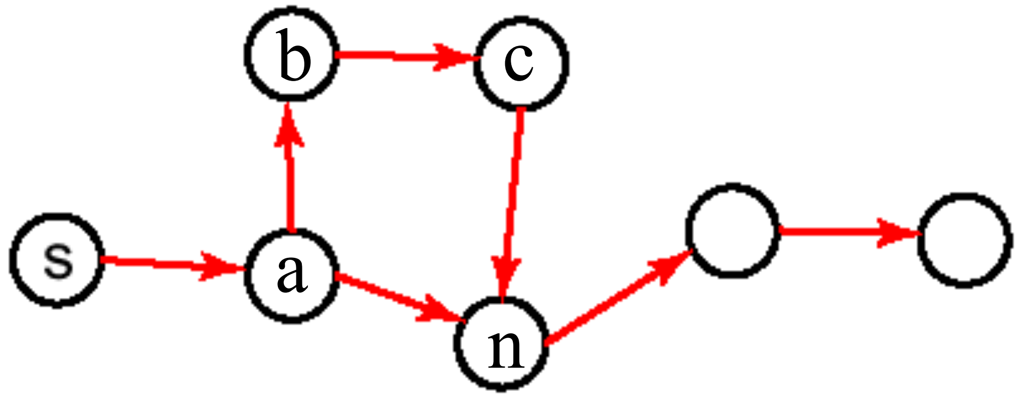
Nodes in this tree correspond to paths in the state space graph

(if multiple start nodes: forest)

May contain cycles!

Cannot contain cycles!

Multiple Path Pruning



- If we only want one path to the solution:
  - Can prune new path p (e.g. sabcn) to node *n* we already reached on a previous path p' (e.g. san)

- To guarantee optimality, either:

- If cost(p) < cost(p')
  - Remove all paths from frontier with prefix p', or
  - Replace prefixes in those paths (replace p' with p)

- Or prove that your algorithm always finds optimal path first

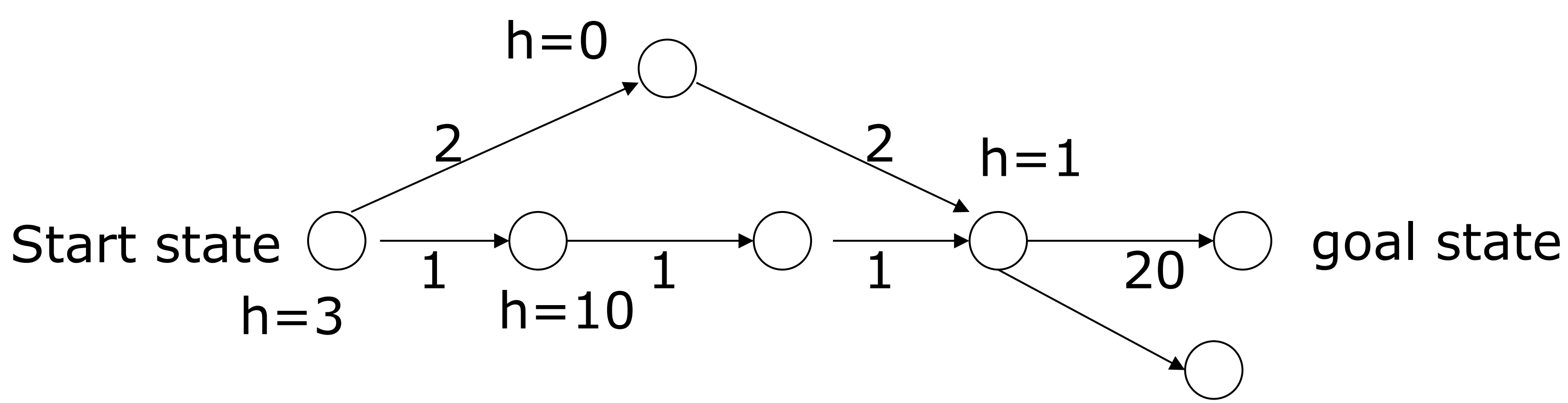
Prove that your algorithm always find the optimal path first

- "Whenever search algorithm A expands a path p ending in node n, this is the lowest-cost path from a start node to n (if all costs ≥ 0)"
  - This is true for

Least Cost Search First | A* | Both of them | None of them

- In general, true only for Least Cost First Search (LCFS)

- Counterexample for A* below: A* expands the upper path first
  - But can recover LCFS's guarantee with monotone heuristic: h is monotone if for all arcs (m,n): $|h(m) - h(n)| \le cost(m,n)$, generalization of admissibility where n is a goal node only with $h(n)=0$.



Iterative Deepening DFS (IDS)

- Depth-bounded depth-first search: DFS on a leash
  - For depth bound d, ignore any paths with longer length

- Progressively increase the depth bound d
  - 1, 2, 3, …, until you find the solution at depth m

- Space complexity: O(bm)
  - At every depth bound, it's just a DFS

- Time complexity: $O(b^m)$
  - Overhead of small depth bounds is very small compared to work at greater depths

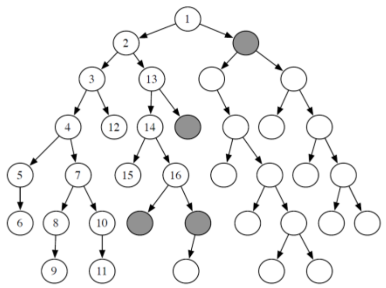
- Optimal: yes
- Complete: yes
- Same idea works for f-value-bounded DFS: IDA*

Lecture Overview

- Recap
- Branch & Bound
- Wrap up of search module
- Constraint Satisfaction Problems (CSPs)

Heuristic DFS

- Other than IDA*, can we use heuristic information in DFS?
  - When we expand a node, we put all its neighbours on the frontier
  - In which order? Matters because DFS uses a LIFO stack
    - Can use heuristic guidance: h or f
    - Perfect heuristic f: would solve problem without any backtracking



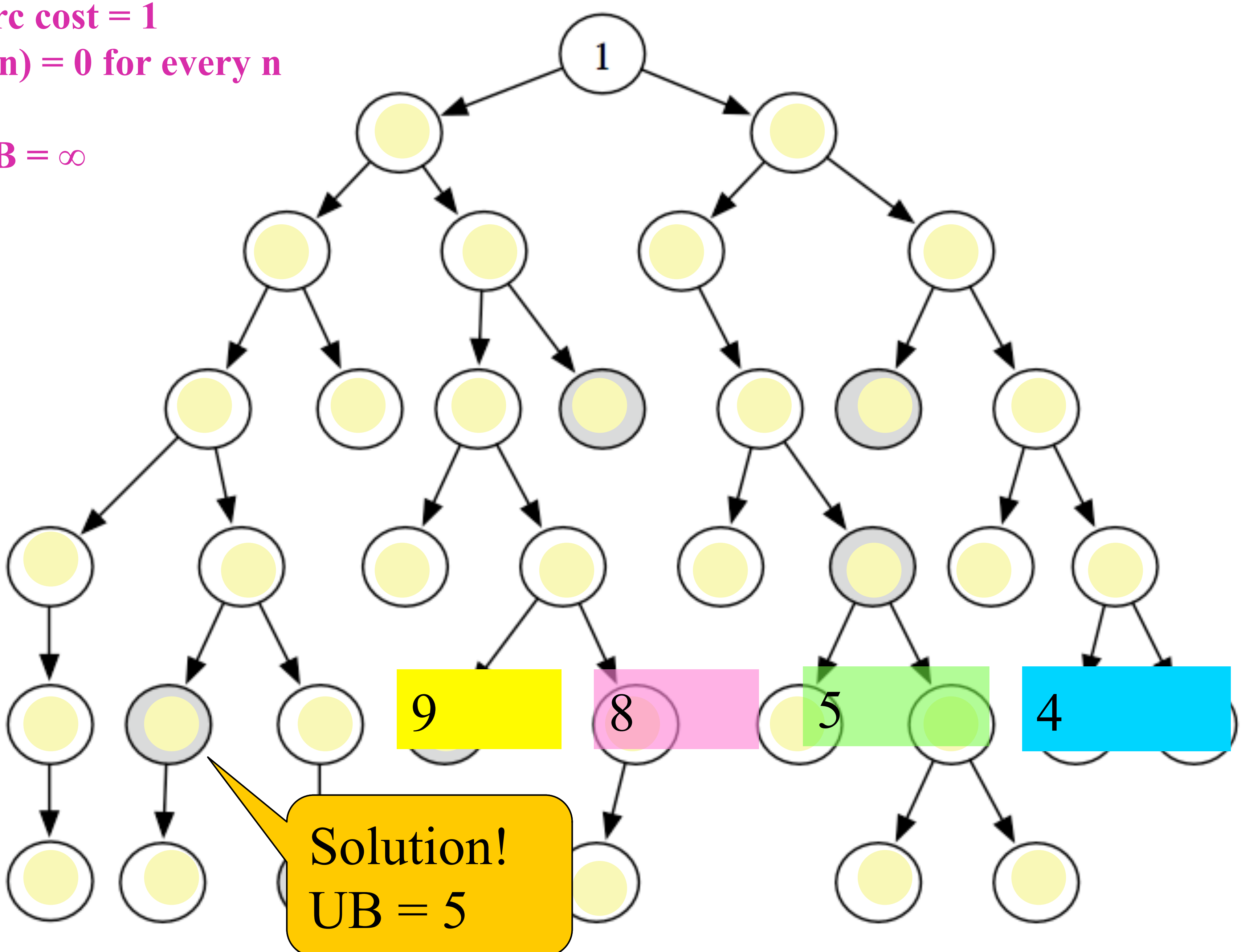
- Heuristic DFS is very frequently used in practice
  - Simply choose promising branches first
  - Based on any kind of information available (no requirement for admissibility)
- Can we combine this with IDA* ? Yes No
  - DFS with an f-value bound (using admissible heuristic h), putting neighbours onto frontier in a smart order (using some heuristic h')
  - Can, of course, also choose h' = h

Branch-and-Bound Search

- One more way to combine DFS with heuristic guidance
- Follows exactly the same search path as depth-first search
  - But to ensure optimality, it does not stop at the first solution found
- It continues, after recording upper bound on solution cost
  - upper bound: *UB* = cost of the best solution found so far
  - Initialized to ∞ or any overestimate of optimal solution cost
- When a path *p* is selected for expansion:
  - Compute lower bound LB(p) = f(p) = cost(p) + h(p)
    - If LB(p) ≥UB, remove *p* from frontier without expanding it (pruning)
    - Else expand *p*, adding all of its neighbors to the frontier
  - Requires admissible h

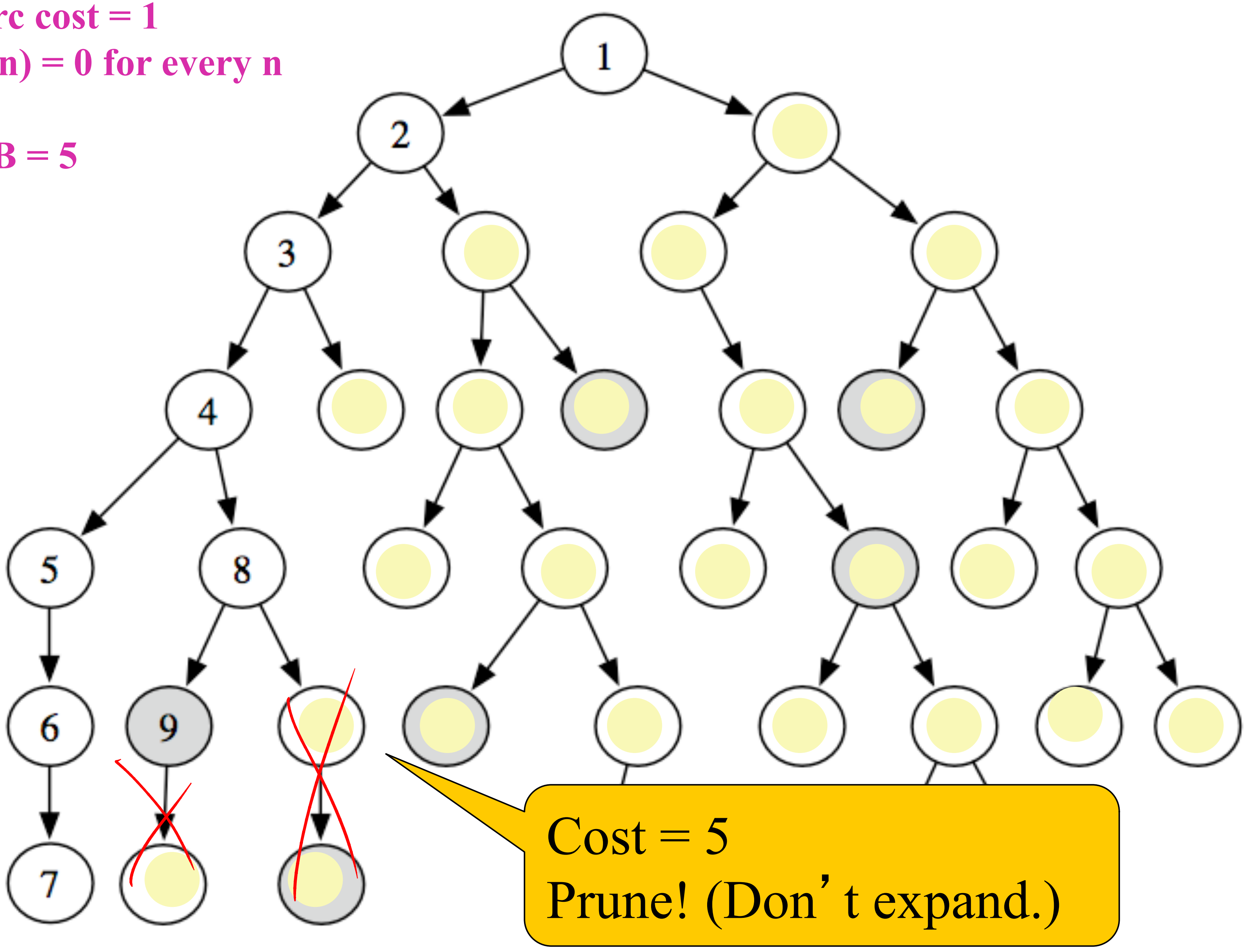
- Arc cost = 1
- h(n) = 0 for every n

- UB = ∞

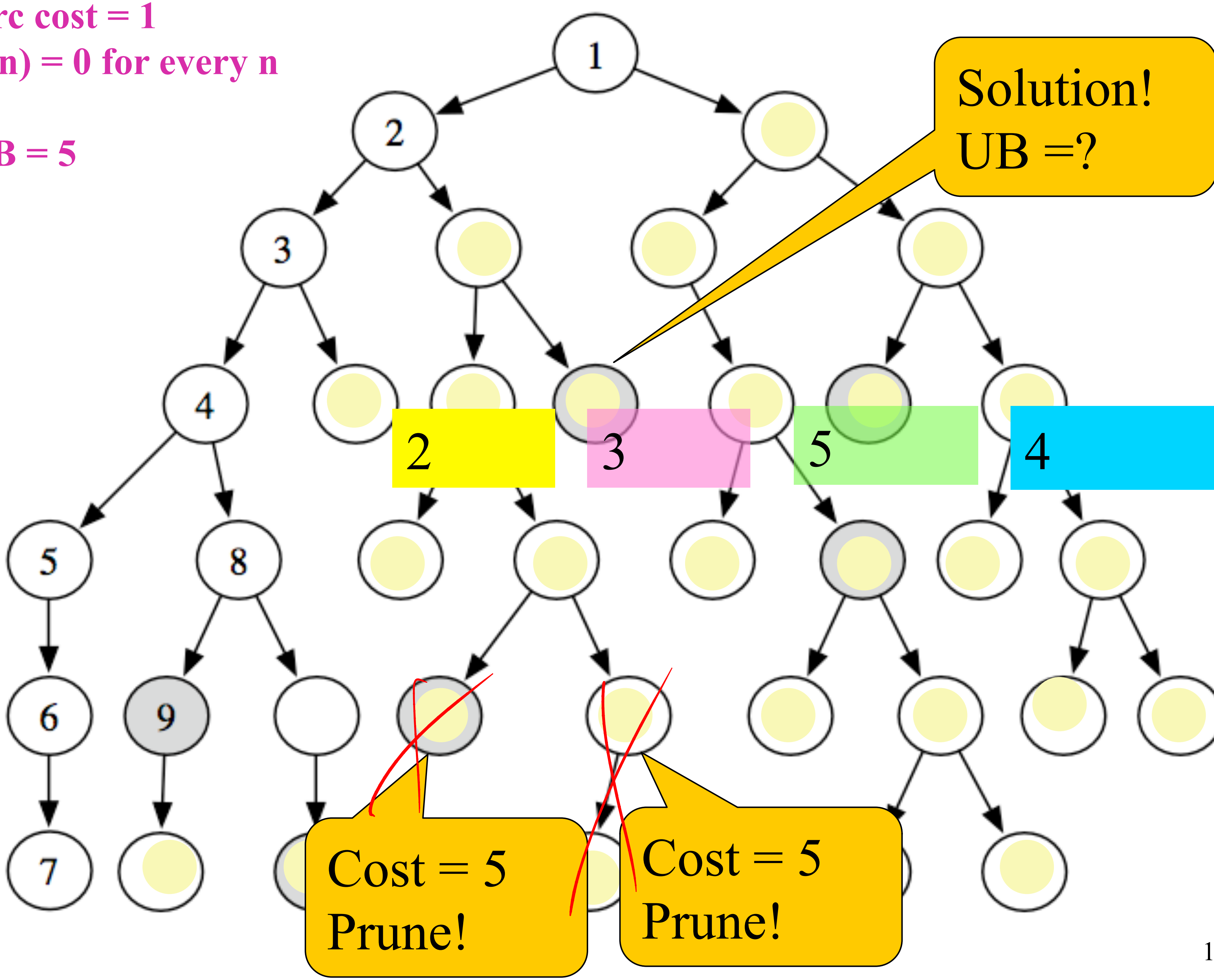


•Arc cost = 1
•h(n) = 0 for every n

•UB = 5

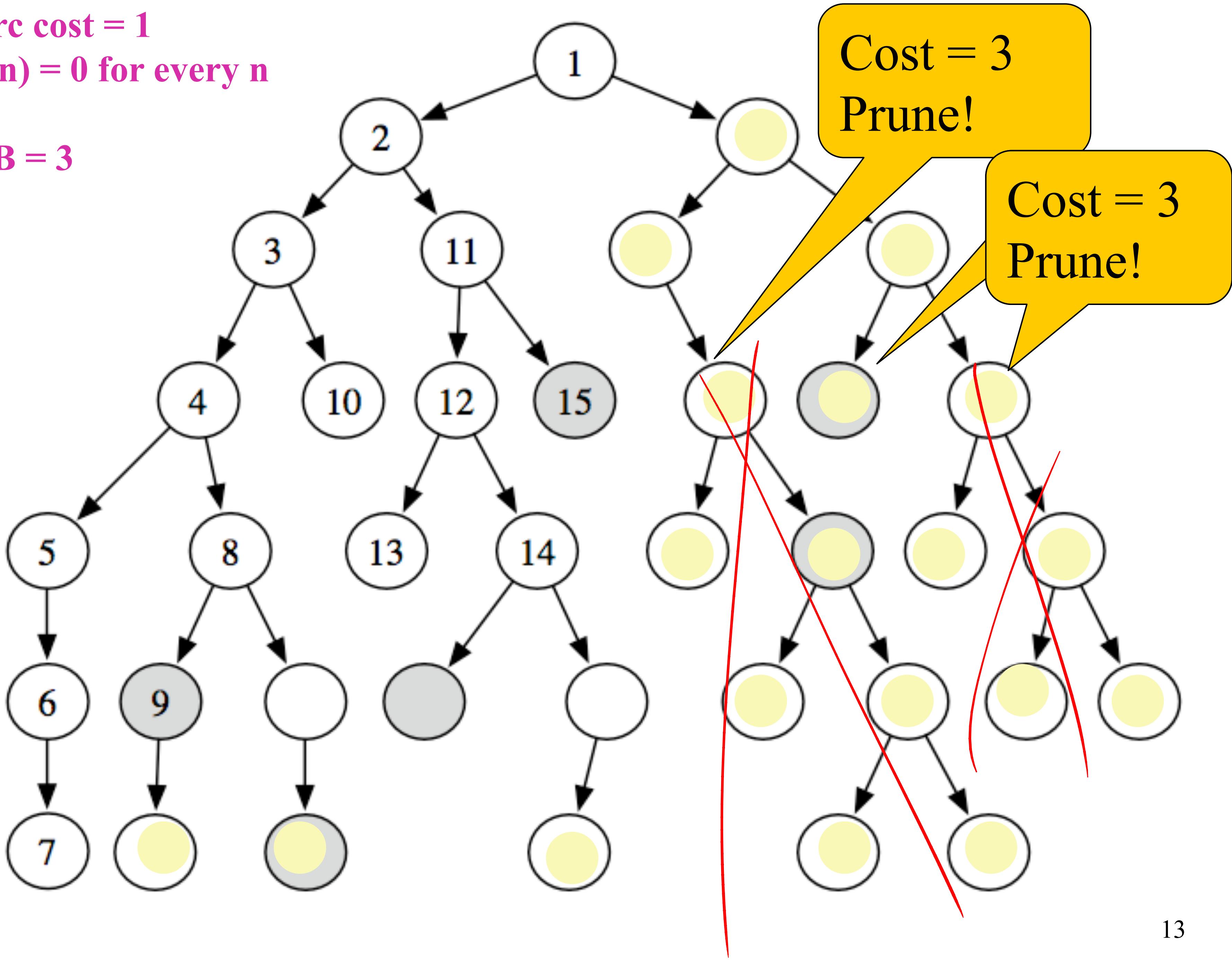


- Arc cost = 1
- h(n) = 0 for every n
- UB = 5



- Arc cost = 1
- h(n) = 0 for every n

- UB = 3



Branch-and-Bound Analysis

- Complete? **YES** **NO** **IT DEPENDS**
  - Same as DFS: can't handle cycles/infinite graphs.
  - But complete if initialized with some finite UB
- Optimal? **YES** **NO** **IT DEPENDS**
  - YES.
- Time complexity: $O(b^m)$
- Space complexity
  - It's a DFS

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

Combining B&B with other schemes

- "Follows the same search path as depth-first search"
  - Let's make that heuristic depth-first search
- Can freely choose order to put neighbours on the stack
  - Could e.g. use a separate heuristic h' that is NOT admissible
- To compute LB(p)
  - Need to compute f value using an admissible heuristic h
- This combination is used a lot in practice

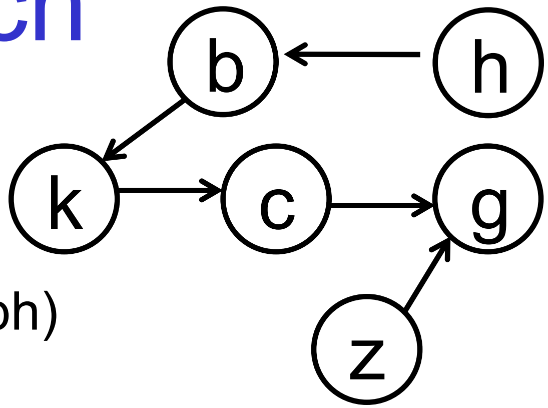
Search methods so far

	Complete	Optimal	Time	Space
DFS	N (Y if no cycles)	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS	Y	Y	$O(b^m)$	$O(mb)$
LCFS (when arc costs available)	Y Costs > 0	Y Costs >=0	$\tilde{O}(b^m)$	$O(b^m)$
Best First (when *h* available)	N	N	$\tilde{O}(b^m)$	$O(b^m)$
A* (when arc costs and *h* available)	Y Costs > 0 *h* admissible	Y Costs >=0 *h* admissible	$\tilde{O}(b^m)$	$O(b^m)$
IDA*	Y (same cond. as A*)	Y	$O(b^m)$	$O(mb)$
Branch & Bound	N (Y if init. with finite UB)	Y	$O(b^m)$	$O(mb)$

Lecture Overview

- Recap
- Branch & Bound
- Wrap up of search module
- Constraint Satisfaction Problems (CSPs)

Direction of Search

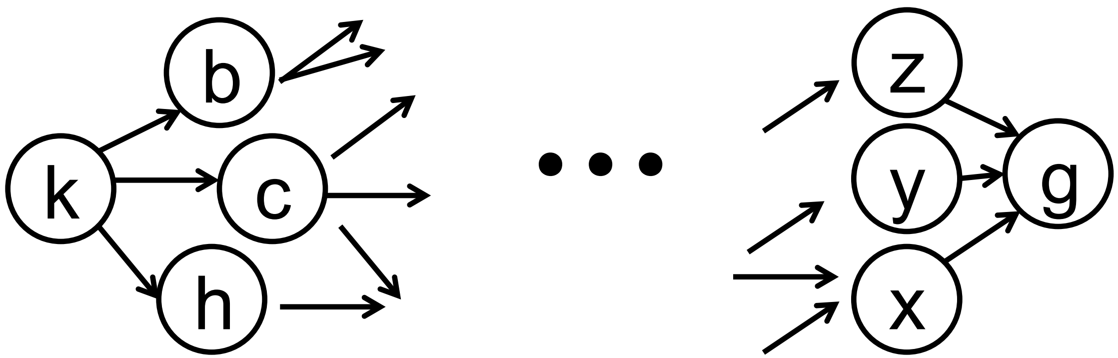


- The definition of searching is symmetric:
  - find path from start nodes to goal node or
  - from goal node to start nodes (in reverse graph)
- Restrictions:
  - This presumes an explicit goal node, not a goal test
  - When the graph is dynamically constructed, it can sometimes be impossible to construct the backwards graph
- Branching factors:
  - Forward branching factor: number of arcs out of a node
  - Backward branching factor: number of arcs into a node
- Search complexity is $O(b^m)$
  - Should use forward search if forward branching factor is less than backward branching factor, and vice versa

Bidirectional search

- You can search backward from the goal and forward from the start simultaneously
  - This wins because $2b^{k/2}$ is much smaller than b^k
  - Can result in exponential savings in time and space

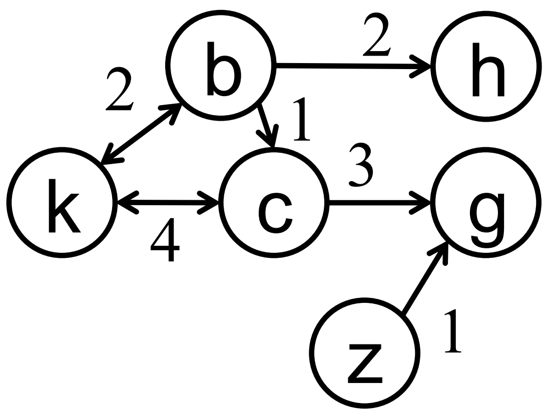
- The main problem is making sure the frontiers meet
  - Often used with one breadth-first method that builds a set of locations that can lead to the goal
  - In the other direction another method can be used to find a path to these interesting locations



Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The actual distance of the shortest path from node n to a goal g

  - dist(g) = 0
  - dist(z) = 1
  - dist(c) = 3
  - dist(b) = 4
  - dist(k) = ? **6** **7** ∞
  - dist(h) = ? **6** **7** ∞



- How could we implement that?
  - Run Dijkstra's algorithm (LCFS with multiple path pruning) in the backwards graph, starting from the goal
- When it's time to act (forward): always pick neighbour with lowest dist value. But you need enough space to store the graph…

Memory-bounded A*

- Iterative deepening A* and B & B use little memory
- What if we have some more memory
  (but not enough for regular A*)?
  - Do A* and keep as much of the frontier in memory as possible
  - When running out of memory
    - delete worst path (highest f value) from frontier
    - Back the path up to a common ancestor
    - Subtree gets regenerated only when all other paths have been shown to be worse than the "forgotten" path

- Complete and optimal if solution is at depth manageable for available memory

Algorithms Often Used in Practice

	Selection	Complete	Optimal	Time	Space
DFS	LIFO	N	N	$O(b^m)$	$O(mb)$
BFS	FIFO	Y	Y	$O(b^m)$	$O(b^m)$
IDS	LIFO	Y	Y	$O(b^m)$	$O(mb)$
LCFS	min cost	Y **	Y **	$\tilde{O}(b^m)$	$O(b^m)$
Best First	min h	N	N	$\tilde{O}(b^m)$	$O(b^m)$
A*	min f	Y**	Y**	$\tilde{O}(b^m)$	$O(b^m)$
B&B	LIFO + pruning	N (Y if UB finite)	Y	$O(b^m)$	$O(mb)$
IDA*	LIFO	Y	Y	$O(b^m)$	$O(mb)$
MBA*	min f	Y**	Y**	$\tilde{O}(b^m)$	$O(b^m)$

** Needs conditions

Learning Goals for search

- **Identify** real world examples that make use of deterministic, goal-driven search agents
- **Assess** the size of the search space of a given search problem.
- **Implement** the generic solution to a search problem.
- **Apply** basic properties of search algorithms:
  - completeness, optimality, time and space complexity
- **Select** the most appropriate search algorithms for specific problems.
- **Define/read/write/trace/debug** different search algorithms
- **Construct** heuristic functions for specific search problems
- **Formally prove** A* optimality.
- **Define optimally** efficient

Learning goals: know how to fill this

	Selection	Complete	Optimal	Time	Space
DFS					
BFS					
IDS					
LCFS					
Best First					
A*					
B&B					
IDA*					

Course Overview

Course Module: *Representation* — Reasoning Technique

Environment

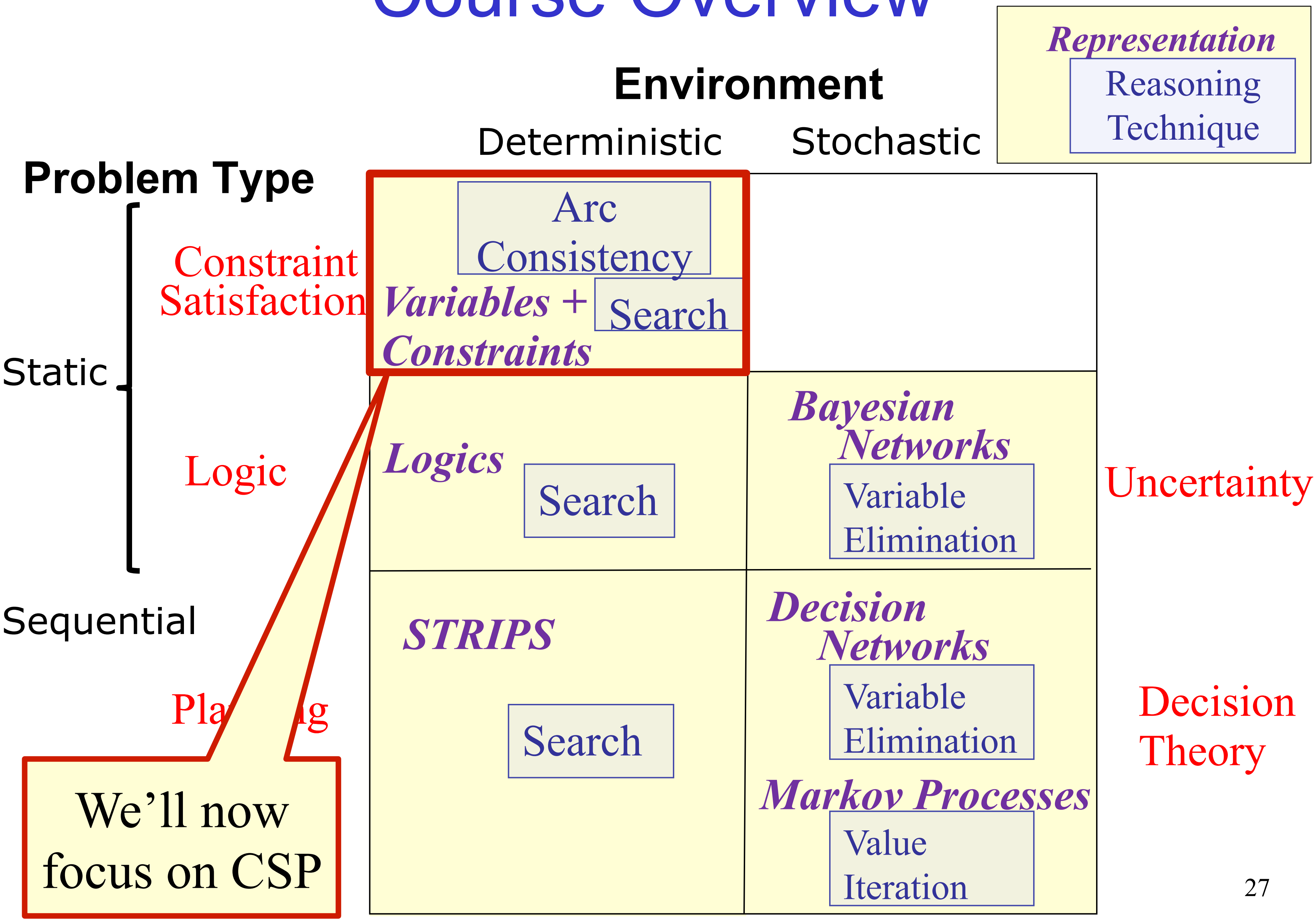
Problem Type		Deterministic	Stochastic	
Static	Constraint Satisfaction	*Variables + Constraints* — Arc Consistency, Search		
Static	Logic	*Logics* — Search	*Bayesian Networks* — Variable Elimination	Uncertainty
Sequential	Planning	*STRIPS* — Search	*Decision Networks* — Variable Elimination; *Markov Processes* — Value Iteration	Decision Theory

Search is everywhere!

Lecture Overview

- Recap
- Branch & Bound
- Wrap up of search module
- Constraint Satisfaction Problems (CSPs)

Course Overview



Main Representational Dimensions (Lecture 2)

Domains can be classified by the following dimensions:

- 1. Uncertainty
  - Deterministic vs. stochastic domains
- 2. How many actions does the agent need to perform?
  - Static vs. sequential domains

An important design choice is:

- 3. Representation scheme
  - Explicit states vs. features (vs. relations)

Explicit State vs. Features (Lecture 2)

How do we model the environment?

- You can enumerate the possible states of the world
- A state can be described in terms of features
  - Assignment to (one or more) variables
  - Often the more natural description
  - 30 binary features can represent 2^{30} =1,073,741,824 states

Variables/Features and Possible Worlds

- Variable: a synonym for feature
  - We denote variables using capital letters
  - Each variable V has a domain dom(V) of possible values
- Variables can be of several main kinds:
  - Boolean: |dom(V)| = 2
  - Finite: |dom(V)| is finite
  - Infinite but discrete: the domain is countably infinite
  - Continuous: e.g., real numbers between 0 and 1
- Possible world
  - Complete assignment of values to each variable
  - In contrast, states also include partial assignments

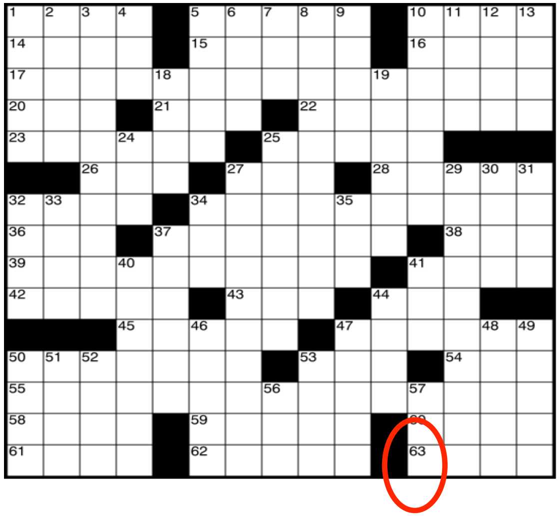
Examples: variables, domains, possible worlds

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of correct length
  - possible worlds: all ways of assigning words

- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells

How many possible worlds?

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of correct length
  - possible worlds: all ways of assigning words

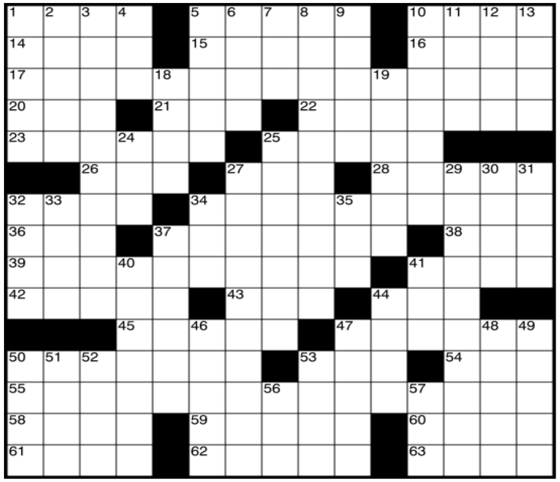


- Number of English words? Let’s say 150,000
  - Of the right length? Assume for simplicity: 15,000 for each word
- Number of words to be filled in? 63
- How many possible worlds? (assume any combination is ok)

$15000*63$ $\quad$ 15000^{63} $\quad$ 63^{15000}

How many possible worlds?

- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells

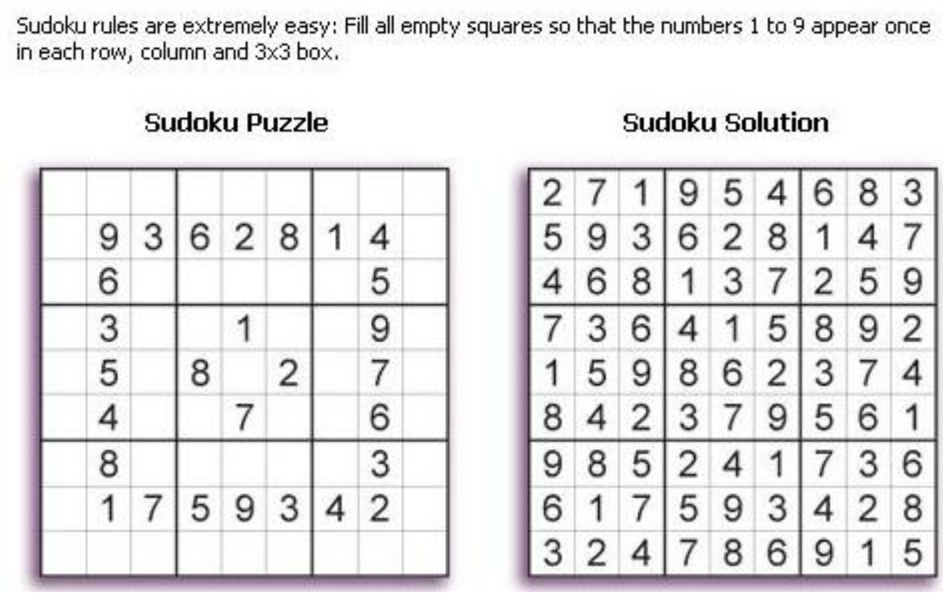


- Number of empty cells? 15*15 – 32 = 193
- Number of letters in the alphabet? 26
-
- How many possible worlds? (assume any combination is ok)

$193*26$ 193^{26} 26^{193}

- In general: (domain size) $^{\#variables}$ (only an upper bound)

Examples: variables, domains, possible worlds


Sudoku rules are extremely easy: Fill all empty squares so that the numbers 1 to 9 appear once in each row, column and 3x3 box.

Sudoku Puzzle

	9	3	6	2	8	1	4	
	6						5	
	3			1			9	
	5		8		2		7	
	4			7			6	
	8						3	
	1	7	5	9	3	4	2	

Sudoku Solution

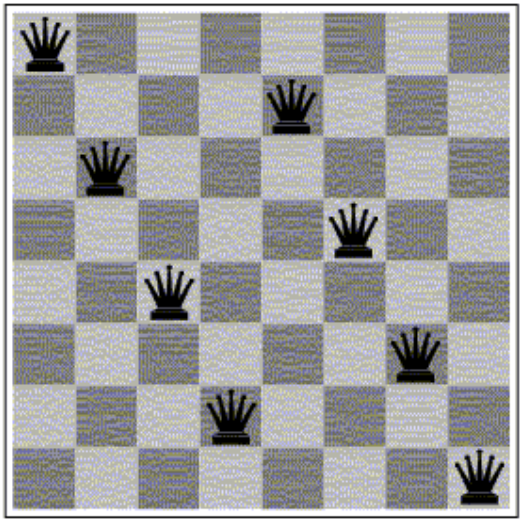
2	7	1	9	5	4	6	8	3
5	9	3	6	2	8	1	4	7
4	6	8	1	3	7	2	5	9
7	3	6	4	1	5	8	9	2
1	5	9	8	6	2	3	7	4
8	4	2	3	7	9	5	6	1
9	8	5	2	4	1	7	3	6
6	1	7	5	9	3	4	2	8
3	2	4	7	8	6	9	1	5

- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - possible worlds: all ways of assigning numbers to cells

Examples: variables, domains, possible worlds

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - possible worlds: time/location assignments for each task

- n-Queens problem
  - variable: location of a queen on a chess board
    - there are n of them in total, hence the name
  - domains: grid coordinates
  - possible worlds: locations of all queens



Constraints

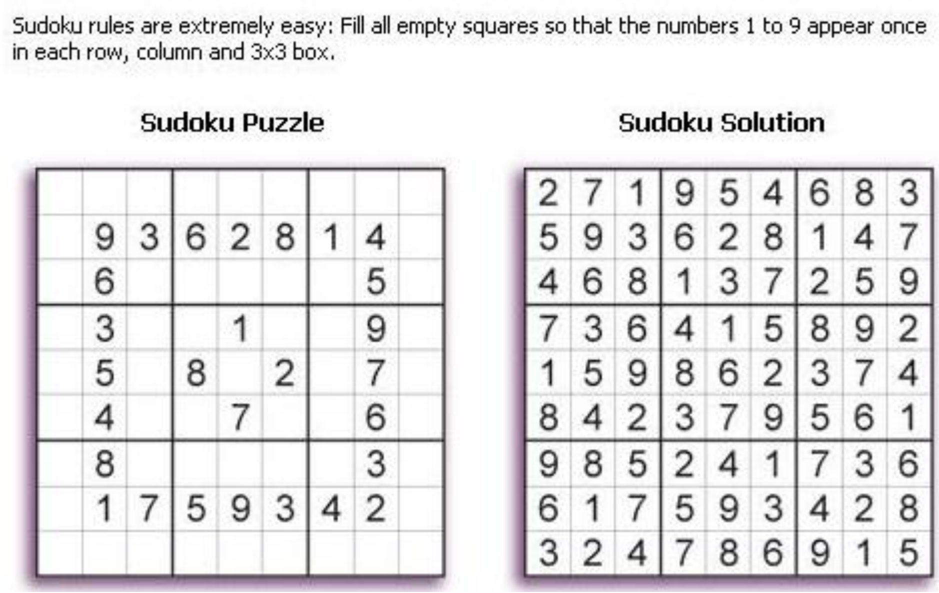
- Constraints are restrictions on the values that one or more variables can take
  - Unary constraint: restriction involving a single variable
    - of course, we could also achieve the same thing by using a smaller domain in the first place
  - k-ary constraint: restriction involving k different variables
    - We will mostly deal with binary constraints
  - Constraints can be specified by
    1. listing all combinations of valid domain values for the variables participating in the constraint
    2. giving a function that returns true when given values for each variable which satisfy the constraint
- A possible world satisfies a set of constraints
  - if the values for the variables involved in each constraint are consistent with that constraint
    1. Elements of the list of valid domain values
    2. Function returns true for those values

Examples: variables, domains, constraints

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of correct length
  - (binary) constraints: two words have the same point where they intersect

- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - (k-ary) constraints: sequences of letters form valid English words

Examples: variables, domains, constraints

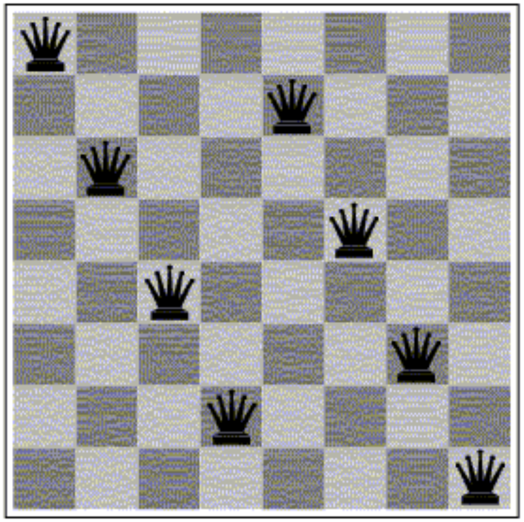


- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - constraints: rows, columns, boxes contain all different numbers

Examples: variables, domains, constraints

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - constraints: tasks can't be scheduled in the same location at the same time; certain tasks can't be scheduled in different locations at the same time; some tasks must come earlier than others; etc.

- n-Queens problem
  - variable: location of a queen on a chess board
    - there are n of them in total, hence the name
  - domains: grid coordinates
  - constraints: no queen can attack another



Constraint Satisfaction Problems: Definition

Definition:
A constraint satisfaction problem (CSP) consists of:
- a set of variables
- a domain for each variable
- a set of constraints

Definition:
A model of a CSP is an assignment of values to all of its variables that satisfies all of its constraints.

Constraint Satisfaction Problems: Variants

- We may want to solve the following problems with a CSP:
  - determine whether or not a model exists
  - find a model
  - find all of the models
  - count the number of models
  - find the best model, given some measure of model quality
    - this is now an optimization problem
  - determine whether some property of the variables holds in all models

Constraint Satisfaction Problems: Game Plan

- Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is NP-hard
  - There is no known algorithm with worst case polynomial runtime
  - We can't hope to find an algorithm that is efficient for all CSPs

- However, we can try to:
  - find consistency algorithms that reduce the size of the search space
  - identify special cases for which algorithms are efficient (polynomial)
  - work on approximation algorithms that can find good solutions quickly, even though they may offer no theoretical guarantees
  - find algorithms that are fast on typical cases

Learning Goals for CSP so far

- Define possible worlds in term of variables and their domains
- Compute number of possible worlds on real examples
- Specify constraints to represent real world problems differentiating between:
  - Unary and k-ary constraints
  - List vs. function format
- Verify whether a possible world satisfies a set of constraints (i.e., whether it is a model, a solution)

- Coming up: CSP as search
  - Read Sections 4.3-2
- Get busy with assignment 1